Pipeline Processor (2)

Hung-Wei Tseng
The “life” of a dynamic instruction?

- Instruction Fetch (IF)
  - Fetch the instruction pointed by PC from memory
- Instruction Decode (ID)
  - Decode the instruction for the desired operation and operands
  - Reading source register values
- Execution (EX)
  - ALU instructions: Perform ALU operations
  - Conditional Branch: Determine the branch outcome (taken/not taken)
  - Memory instructions: Determine the effective address for data memory access
- Data Memory Access (MEM) — Read/write data memory
- Write Back (WB) — Present ALU result/read value in the target register
- Update PC
  - If the branch is taken — set to the branch target address
  - Otherwise — advance to the next instruction — current PC + 4
Recap: Single-cycle processor

Instruction Fetch
- PC
- Instruction Memory [31:0]
  - Instruction Address [31:0]
  - Instruction [31:0]
  - Instruction Fetch
  - Instruction Decode, prepare operands
  - Next PC

Instruction Decode, prepare operands
- Instruction [31-26]
- RegDst
- MemToReg
- MemRead
- MemWrite
- ALUOp
- ALUSrc
- RegWrite
- Control
- Next PC

Register File
- Read Reg 1
- Read Reg 2
- Write Reg
- Write Data
- Write Back

ALU
- ALU Op
- ALU Ctrl.
- Is zero?
- ALUSrc
- MemoryRead
- MemoryWrite
- RegWrite

Data Memory
- Data Address [31:0]
- Data [31:0]
- Is zero?

Write Back
- MemoryWrite
- Write Data [31:0]
- Data Memory
- Access

Execute
- ALU
- ALU Op
- ALU Ctrl.
- Is zero?
- ALUSrc
- MemoryRead
- MemoryWrite
- RegWrite
Performance of a single-cycle processor

• How many of the following statements about a single-cycle processor is correct?

① The CPI of a single-cycle processor is always 1
② If the single-cycle implements MIPS ISA, the memory instruction will determine the cycle time
③ Hardware elements are mostly idle during a cycle
④ We can always reduce the cycle time of a single-cycle processor by supporting fewer instructions — Only if this instruction is the most time-critical one

A. 0
B. 1
C. 2
D. 3
E. 4
Recap: Pipelining

After this point, we are completing an instruction each cycle!

All hardware parts are in use

Cycles
Instruction = 1
Recap: Single-cycle processor

Instruction Memory

Instruction Address [31:0] Instruction [31:0]

Control

RegDst MemToReg Branch MemoryRead MemoryWrite ALUOp ALUSrc RegWrite

Instruction [31-26]

Instruction [25-21]

Instruction [20-16]

Instruction [15-11]

Instruction [15-0]

Instruction [5-0]

Instruction [31-21]

Instruction [20-11]

Instruction [15-0]

Instruction [5-0]

Register File

Read Reg 1 Read Data 1 Read Reg 2 Read Data 2 Write Reg Read Data 2 Write Data

ALU

Data Memory

Data Address [31:0] Write Data[31:0]

Is zero?

ALU Ctrl.

0 1 0 1

shift left 2

PCSrc
Recap: Pipelined processor
add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9, $10, $11
sw $1, 0($12)
Recap: Tips of drawing a pipeline diagram

• Each instruction has to go through all 5 pipeline stages: IF, ID, EXE, MEM, WB in order — only valid if it’s single-issue, MIPS 5-stage pipeline

• An instruction can enter the next pipeline stage in the next cycle if
  • No other instruction is occupying the next stage
  • This instruction has completed its own work in the current stage
  • The next stage has all its inputs ready and it can retrieve those inputs

• Fetch a new instruction only if
  • We know the next PC to fetch
  • We can predict the next PC
  • Flush an instruction if the branch resolution says it’s mis-predicted.
Recap: Pipelining

After this point, we are completing an instruction each cycle!

Cycles
Instruction = 1

add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9, $10, $11
sw $1, 0($12)
xor $13, $14, $15
and $16, $17, $18
add $19, $20, $21
sub $22, $23, $24
lw $25, 4($26)
sw $27, 0($28)
Limitations of pipelining

- How many of the following descriptions about pipelining is correct?
  - You can always divide stages into short stages with latches to improve performance  
    - Only if this stage is the most time-critical one
  ② Pipeline registers incur overhead for each pipeline stage
  ③ The latency of executing an instruction in a pipeline processor is longer than a single-cycle processor  
    - You have pipeline registers and each stage needs to be equally long
  ④ The throughput of a pipeline processor is usually better than a single-cycle processor

A. 0
B. 1
C. 2
D. 3
E. 4
Programming languages

- How many instructions are there in "Hello, world!"

<table>
<thead>
<tr>
<th>Programming language</th>
<th>Instruction count</th>
<th>LOC</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>600k</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C++</td>
<td>3M</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Java</td>
<td>~210M</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Perl</td>
<td>10M</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Python</td>
<td>~30M</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Both version A and B produces the same output. Without compiler optimization, which version of code would have significantly better performance?

A. Version A
B. Version B
C. They are about the same (less than 10% difference)

— Because we have pipelined instructions, the CPI of one instruction doesn’t matter as long as we can keep the pipeline busy
Outline

• The 5-stage MIPS Pipeline processor
• Pipeline Hazards
The 5-stage MIPS Pipeline Processor
### Can we get them right?

- Given a simple pipelined MIPS processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
</tr>
<tr>
<td>b</td>
<td>lw $4, 0($1)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
</tr>
<tr>
<td>c</td>
<td>sub $6, $7, $8</td>
<td>sub $6, $7, $8</td>
<td>bne $0, $7, L</td>
<td>sub $6, $7, $8</td>
</tr>
<tr>
<td>d</td>
<td>sub $9, $10, $11</td>
<td>sub $9, $1, $10</td>
<td>sub $9, $10, $11</td>
<td>sub $9, $10, $11</td>
</tr>
<tr>
<td>e</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
</tr>
</tbody>
</table>

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Given a simple pipelined MIPS processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
</tr>
<tr>
<td>b</td>
<td>lw $4, 0($1)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
</tr>
<tr>
<td>c</td>
<td>sub $6, $7, $8</td>
<td>sub $6, $7, $8</td>
<td>bne $0, $7, L</td>
<td>sub $6, $7, $8</td>
</tr>
<tr>
<td>d</td>
<td>sub $9, $10, $11</td>
<td>sub $9, $1, $10</td>
<td>sw $11, 0($12)</td>
<td>sub $9, $10, $11</td>
</tr>
<tr>
<td>e</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
</tr>
</tbody>
</table>

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Draw the pipeline diagrams

The desired value of $x_1$ is not ready yet.

Both instructions need $x_1$.

Doesn't know what to fetch at this moment.
Can we get them right?

Given a simple pipelined MIPS processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

A. 0
B. 1
C. 2
D. 3
E. 4

- b cannot get x1 produced by a before WB
- both a and d are accessing x1 at the 5th cycle
- We don’t know if d & e will be executed or not until c finishes
Pipeline hazards
Three pipeline hazards

- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that’s not yet generated or propagated when the instruction needs that
Can we get them right?

- Given a simple pipelined MIPS processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
</tr>
<tr>
<td>b</td>
<td>lw $4, 0($1)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
<td>lw $4, 0($5)</td>
</tr>
<tr>
<td>c</td>
<td>sub $6, $7, $8</td>
<td>sub $6, $7, $8</td>
<td>sub $6, $7, $8</td>
<td>sub $6, $7, $8</td>
</tr>
<tr>
<td>d</td>
<td>sub $9, $10, $11</td>
<td>sub $9, $1, $10</td>
<td>add $1, $2, $3</td>
<td>add $1, $2, $3</td>
</tr>
<tr>
<td>e</td>
<td>sw $1, 0($12)</td>
<td>sw $11, 0($12)</td>
<td>sw $1, 0($12)</td>
<td>sw $1, 0($12)</td>
</tr>
</tbody>
</table>

A. 0
B. 1
C. 2
D. 3
E. 4

- **Data Hazard:** b cannot get $x_1$ produced by a before WB
- **Structural Hazard:** bne $0, $7, L
- **Control Hazard:** We don’t know if d & e will be executed or not until c finishes
Structural Hazards
Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Solution: insert no-ops (e.g, add x0, x0, x0) between them
- Drawback
  - If the number of pipeline stages changes, the code won’t work
  - Slow

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
add $0, $0, $0
sub $9, $1, $10
sw  $11, 0($12)
```
Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Solution: stall the later instruction, allowing the write to present the change in the register and the later can get the desired value
- Drawback: slow

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9, $1, $10
sw  $11, 0($12)
```
Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Better solution: write early, read late
  - Writes occur at the clock edge and complete long enough before the end of the clock cycle.
  - This leaves enough time for outputs to settle for reads
  - The revised register file is the default one from now!

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9, $1, $10
sw  $11, 0($12)
```
What pair of instructions will be problematic if we allow ALU instructions to skip the "MEM" stage?

A. a & b
B. a & c
C. b & e
D. c & e
E. None

Structural Hazards

a: lw $1, 0($2)
b: add $3, $4, $5
c: sub $6, $7, $8
d: sub $9, $10, $11
e: sw $1, 0($12)
Structural Hazards

• What pair of instructions will be problematic if we allow ALU instructions to skip the “MEM” stage?
  a: lw $1, 0($2)
  b: add $3, $4, $5
  c: sub $6, $7, $8
  d: sub $9,$10,$11
  e: sw $1, 0($12)
  A. a & b
  B. a & c
  C. b & e
  D. c & e
  E. None
Structural Hazards

What pair of instructions will be problematic if we allow ALU instructions to skip the “MEM” stage?

A: \text{l} \text{w} $1, 0(\$2)$ and \text{add} $3, 4, 5$
B: \text{add} $3, 4, 5$ and \text{sub} $6, 7, 8$
C: \text{sub} $6, 7, 8$ and \text{sw} $1, 0(\$12)$
D: \text{sub} $9, 10, 11$ and \text{sw} $1, 0(\$12)$
E: None

A. a & b
B. a & c
C. b & e
D. c & e
E. None
Structural Hazards

• Stall can address the issue — but slow
• Compiler optimization — what if the hardware design changes?
• Improve the pipeline unit design to allow parallel execution
Which version is faster?

Both version A and B produces the same output. Without compiler optimization, which version of code would have significantly better performance?

A. Version A
B. Version B
C. They are about the same (less than 10% difference)

— Because we have pipelined instructions, the CPI of one instruction doesn’t matter as long as we can keep the pipeline busy.
Both version A and B produces the same output. Without compiler optimization, which version of code would have significantly better performance?

A. Version A  
B. Version B  
C. They are about the same (less than 10% difference)
• Both version A and B produces the same output. Without compiler optimization, which version of code would have significantly better performance?
  A. Version A
  B. Version B
  C. They are about the same (less than 10% difference)
Data hazards
Data hazards

- An instruction currently in the pipeline cannot receive the "logically" correct value for execution
- Data dependencies
  - The output of an instruction is the input of a later instruction
  - May result in data hazard if the later instruction that consumes the result is still in the pipeline
Example: vector scaling

```
i = 0;
do {
    vector[i] += scale;
} while ( ++i < size )
```

```
sll $t0,$a1, 2
add $t1,$t0,$a0
lw $t0,0($a0)
add $t0,$t0, $t2
sw $t0,0($a0)
addi $a0,$a0, 4
bne $a0,$t1, LOOP
```
How many dependencies do we have?

- How many pairs of data dependences are there in the following MIPS instructions?

```
lw    $t0,0($a0)
add   $t0,$t0, $t2
sw    $t0,0($a0)
addi  $a0,$a0, 4
bne   $a0,$t1, LOOP
```

A. 1
B. 2
C. 3
D. 4
E. 5
How many dependencies do we have?

- How many pairs of data dependences are there in the following MIPS instructions?

```
lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP
```

A. 1
B. 2
C. 3
D. 4
E. 5
How many pairs of data dependences are there in the following MIPS instructions?

lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP

A. 1  
B. 2  
C. 3  
D. 4  
E. 5
Solution 1: Let’s try “stall” again

- Whenever the input is not ready when the consumer is decoding, just stall — the consumer stays at ID.
How many of data hazards?

- How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline?

```assembly
lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP
```

A. 1  
B. 2  
C. 3  
D. 4  
E. 5
How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline?

```
lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP
```

A. 1
B. 2
C. 3
D. 4
E. 5
How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline?

```
lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP
```

A. 1  
B. 2  
C. 3  
D. 4  
E. 5
Solution 2: Data forwarding

• Add logics/wires to forward the desired values to the demanding instructions

• In our five stage pipeline — if the instruction entering the EXE stage consumes a result from a previous instruction that is entering MEM stage or WB stage
  • A source of the instruction entering EXE stage is the destination of an instruction entering MEM/WB stage
  • The previous instruction must be an instruction that updates register file
Pipelined processor with Data Forwarding
How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline with "full" data forwarding?

```
lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP
```

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline with “full” data forwarding?

lw   $t0,0($a0)
add  $t0,$t0, $t2
sw   $t0,0($a0)
addi $a0,$a0, 4
bne  $a0,$t1, LOOP

A. 0  
B. 1  
C. 2  
D. 3  
E. 4
Do we still have to stall?

- How many pairs of instructions in the following MIPS instructions will result in data hazards/stalls in a basic 5-stage MIPS pipeline with "full" data forwarding?

lw $t0,0($a0)
add $t0,$t0, $t2
sw $t0,0($a0)
addi $a0,$a0, 4
bne $a0,$t1, LOOP

A. 0
B. 1
C. 2
D. 3
E. 4
Pipelined processor with Data Forwarding

Instruction Memory
- Instruction Address [31:0]
- Instruction [31:0]

Register File
- Read Reg 1
- Read Data 1
- Read Reg 2
- Read Data 2
- Write Reg
- Write Data

Forwarding
- ForwardA
- ForwardB

Data Memory
- Data Address [31:0]
- Data[31:0]

Instruction
- Instruction [31-26]
- Instruction [25-21]
- Instruction [20-16]
- Instruction [15-11]
- Instruction [15-0]

Instruction
- Instruction [5-0]

Control
- RegDst
- MemToReg
- Branch
- MemoryRead
- MemoryWrite
- ALUOp
- ALUSrc
- RegWrite

ALU
- Is zero?
- sign-extend
- ALU Ctrl.

Hazard Detection
- Hazard Detection

PCWrite
- PCWrite

IF/ID
- IF/IDWrite
- Instruction

ID/EX
- ID/EX MemoryRead
- ID/EX Write

EX/MEM
- EX/MEM MemoryRead
- EX/MEM Write

MEM/WB
- MEM/WB RegisterRead
- MEM/WB Write
**Announcements**

- Plan your time carefully next week!
  - Reading quiz — due tomorrow
  - Assignment #2 — due next Monday
  - Assignment #3 — due next Wednesday
  - Midterm
    - Next Thursday during the lecture — make sure that you’re available, we will use Zoom
    - The lecture on Wednesday will provide a review, highlight, sample midterm

- Resources
  - Ask questions — piazza
  - Reading quizzes, turning in assignments — Canvas
  - Slides, schedule, assignment questions — Check our website
  - Video archive — Prof. Usagi’s Youtube channel
Computer Science & Engineering